

```
// LEDs - maybe declare subsets and allocate each FPGA some
// great care has to be taken if both FPGAs try to access the same LEDs
////////////////////////////////////
macro expr LED_pins = {data = {  "AU13", "AT14", "AV12", "AU14",
5                                "AW12", "AT15", "AV13",
                                "AU15"}};

10
////////////////////////////////////
// ATA Interface
////////////////////////////////////
macro expr ATA_pins = {data = {  "AU26", "AV27", "AT26", "AW28", "AU27",
15                                "AV28", "AW29", "AT27",
                                "AW30", "AU28",
                                "AV30", "AV29", "AW31",
                                "AU29", "AV31",
                                "AT29", "AW32", "AU30",
20                                "AW33", "AT30",
                                "AV33", "AU31", "AT31",
                                "AW34", "AV32",
                                "AV34", "AU32", "AW35",
                                "AT32", "AV35",
25                                "AU33", "AW36",
                                "AT33"}};
```

```
////////////////////////////////////
// Expansion Bus (32 bits)
5  //////////////////////////////////////

macro expr E_pins = {data = {      "AV17", "AU18", "AW17", "AT19", "AV18",
                                   "AU19", "AW18", "AU21",
                                   "AV19", "AW20",
10  "AV20", "AR22", "AV23",
                                   "AW21", "AU23",
                                   "AV21", "AT23", "AW22",
                                   "AR23", "AV22",
                                   "AV24", "AW23",
15  "AW24", "AU24", "AW25",
                                   "AT24", "AV25", "AU25",
                                   "AW26", "AT25",
                                   "AV26", "AW27"}}};

20

////////////////////////////////////
// Serial H Bus
////////////////////////////////////

macro expr SERIALH_pins = {data = {"F39", "H37", "F38", "H36", "E39", "G37",
25  "E38"}}};

////////////////////////////////////
```

// SelectLink Bus - Directly connects the 2 FPGAs

////////////////////////////////////

```
macro expr SL_pins = {data = {    "AV3", "AU4", "AV5", "AT6", "AV4", "AU6",
5                                "AW4", "AT7", "AW5",
                                "AU7", "AV6", "AT8",
                                "AW6", "AU8", "AV7",
                                "AT9", "AW7", "AV8",
                                "AU9", "AW8", "AT10",
10    "AV9", "AU10", "AW9",
                                "AT11", "AV10", "AU11",
                                "AW10", "AU12", "AV11",
                                "AT13", "AW11"}}};
```

////////////////////////////////////

//VGA interface

////////////////////////////////////

```
macro expr VGA_pins = {data = {    "AW13", "AV14", "AT16", "AW14", "AU16",
20                                "AV15", "AR17", "AW15",
                                "AT17", "AU17",
                                "AV16", "AR18", "AW16",
25    "AT18"}}};
```

macro expr vga_vsync_pin = { data = { "AV14" } };

macro expr vga_hsync_pin = { data = { "AW13" } };

macro expr vga_data_pins = { data = { "AT16", "AW14", "AU16", "AV15",

POSSIBLE REGIONS

```
"AR17", "AW15", "AT17", "AU17",  
"AV16", "AR18", "AW16", "AT18"} };
```

```
5 // macros for compatibility with existing programs  
macro expr vsync_pin = { "AV14" };  
macro expr hsync_pin = { "AW13" };  
macro expr video_spec = { data = { "AT16", "AW14", "AU16", "AV15",  
10 "AR17", "AW15", "AT17", "AU17",  
"AV16", "AR18", "AW16", "AT18"} };  
  
////////////////////////////////////  
// CPLD interface pins  
15 //////////////////////////////////////  
  
macro expr BUSMaster_pin = { data = { "C26" } }; // P12  
macro expr FPcom_pins = { data = { "B26", "C27", "A27" } }; //P14 P15 P16  
  
20 //////////////////////////////////////  
// Serial Ports pins  
////////////////////////////////////  
  
25 macro expr SERIAL_pins = {data = {"AV36", "AU34", "AU36", "AT34"}};  
  
macro expr rs232_txd_pin = {data = { "AV36"}};  
macro expr rs232_rxd_pin = {data = { "AU36"}};
```

```
macro expr rs232_rts_pin = {data = { "AU34"}};
macro expr rs232_cts_pin = {data = { "AT34"}};
```

```
5  //////////////////////////////////////
   // USB
   //////////////////////////////////////
```

```
macro expr USBMaster_pin = { data = { "D26" }}; // P13
```

```
10 macro expr USBD_pins = {data = {"C29", "A30", "D29", "B30", "C30", "A31", "D30",
   "A32"}};
```

```
macro expr USBMS_pins = { data = {"D27"} };
```

```
15 macro expr USBnRST_pins = { data = {"B27"} };
```

```
macro expr USBIRQ_pins = { data = {"C28"} };
```

```
20 macro expr USBA0_pins = { data = {"A28"} };
```

```
macro expr USBnRD_pins = { data = {"B28"} };
```

```
macro expr USBnWR_pins = { data = {"B29"} };
```

```
25 macro expr USBnCS_pins = { data = {"A29"} };
```

```
#endif _KOMPRESSOR_SLAVE_HEADER
```

to the file

Appendix C

Following is a description of a parallel port interface that gives full access to the all the parallel port pins and implements a parallel port data transfer functionality that can be used in conjunction with the ESL download utility

```
5 // *****
// Parallel port controller
// *****

10 // Instantiates a component that controls the parallel port.
// This is to be run in parallel in the main loop. The interfaces
// provide the user with abstracts to use deal efficiently with the
// component.

15 // *****
// Interfaces
//
// API to Parallel Port - for direct access to the pins

20 //
// PpWriteData((unsigned 8)byte) -- write byte to data pins
// PpReadData((unsigned 8)byte) -- read byte from data pins
// PpReadControl((unsigned 4)control_port) -- read the control port
// PpReadStatus((unsigned 6)status_port) -- read the status port
25 // PpSetStatus((unsigned 6) status_port) -- write to the status port
//
//
// API for the ESL parallel data transfer utility
//
```

```
// OpenPP(error) -- open the parallel port for data transfer
// ClosePP(error) -- close the port
// SetSendMode(error) -- set the port to send mode
// SetRecvMode(error) -- set the port to receive mode
5 // SendPP(byte, error) -- send a byte over the port
// ReadPP(byte, error) -- read a byte from the port
//
// error returns the result of the command:
// 0 - no error
10 // 1 - buffer error
// 2 - timeout error
//
// Note: SendPP and ReadPP will block the thread until a byte is transmitted or the
// timeout
15 // value is reached. If you need to do some processing while waiting for a
// communication
// use a 'pralt' statement to read from the global pp_recv_chan channel or write to the
// pp_send_chan channel.
20
//
// The Nitty Gritty
//
25
// The necessary channels
chan unsigned 8 pp_send_chan, pp_recv_chan;
chan unsigned 2 pp_command, pp_error;
```



```
chan pp_data_send_channel, pp_data_read_channel, pp_control_port_read;  
chan pp_status_port_read, pp_status_port_write;
```

```
5  #define OPEN_CHANNEL  0  
   #define CLOSE_CHANNEL 1  
   #define SEND_MODE      2  
   #define RECV_MODE      3  
  
10 #define PP_NO_ERROR      0  
   #define PP_HOST_BUFFER_NOT_FINISHED  1  
   #define PP_OPEN_TIMEOUT 2
```

```
   // Currently the functions don't act on any errors, but this can easily be added if  
15 required.  
   // return of error code could also be used to generate a time-out condition.
```

```
macro proc OpenPP(error)  
20 {  
    pp_command ! OPEN_CHANNEL;  
    pp_error ? error;  
}  
  
25  
macro proc ClosePP(error)  
{  
    pp_command ! CLOSE_CHANNEL;  
    pp_error ? error;
```

FORTRAN-012901-452460

}

macro proc SetSendMode(error)

{

5 pp_command ! SEND_MODE;

 pp_error ? error;

}

macro proc SetRecvMode(error)

10 {

 pp_command ! RECV_MODE;

 pp_error ? error;

}

15

macro proc WritePP(byte, error)

{

 pp_send_chan ! byte;

}

20

macro proc ReadPP(byte, error)

{

 pp_recv_chan ? byte;

25 }

// *****

```
// Parallel port controller
// *****

// Host Channel Control (HCC)  nAutoFeed
5 // FPGA Channel Control (FCC)  DONE
// Host Data Control (HDC)      nSelect_in
// FPGA Data Control (FDC)      nACK
// FPGA ready to communicate (FRTC) PE

10 // HCC indicates that host is sending - end of the buffer
// FCC controls direction of communication
// FRTC indicates that FPGA is ready
// when FPGA sets FCC low, rising edge on FDC when data applied
15 // lower when host responds with HDC high
// when FCC high FPGA is in receive mode and host applies data
// on rising edge on HDC. FPGA responds with FDC high and host
// then lowers HDC. Host will keep data byte on pins till FDC is
// lowered again by the FPGA

20 // chan unsigned 8 pp_data_chan;
// chan unsigned 4 pp_control_chan;
// chan unsigned 5 pp_status_chan;

25

////////////////////////////////////
// Macro to implement ESLs bi-directional host-fpga
// data transfer protocol
```

```
// Accesses the physical layer
////////////////////////////////////
```

5

```
macro proc Test_PP()
{
```

10

```
    unsigned 4 control_port;
    unsigned 6 status_port;

    unsigned 21 counter;
```

15

```
    // PpSetControl(0b0000);
    PpSetStatus(0b000000);
```

20

```
    do
    {
        counter++;
    }while(counter != 0);
```

25

```
    PpSetStatus(0b000001);

    do
    {
        counter++;
    }while(counter != 0);

    PpSetStatus(0b000010);
```

```
do
{
counter++;
}while(counter != 0);

PpSetStatus(0b000100);

do
{
counter++;
}while(counter != 0);

PpSetStatus(0b001000);

do
{
counter++;
}while(counter != 0);

PpSetStatus(0b010000);

do
{
counter++;
}while(counter != 0);
```

```
PpSetStatus(0b000000);

    do
5      {
        counter++;
      }while(counter != 0);

    PpSetStatus(0b011111);
10
    while(1)
    {
        PpReadControl(debug_control);
    }
15 }

20

macro proc pp_coms(pp_send_chan, pp_recv_chan, pp_command, pp_error)
{
25
    // bit masks for accessing control and status ports

    //control_port = nSelect_in.in @ init.in @ nAutofeed.in @ nStrobe.in;
    #define HCC control_port[1] //0b0010 //nAutofeed pin on control port
```

```
#define HDC control_port[2] //0b0100 //nInit pin on control port

//status_port = ppsdir @ busy @ nAck @ pe @ select @ nError;
#define FRTC 0b000010 //pe pin on status port
5 #define FCC 0b000100 //select pin on status port
#define FDC 0b001000 //nAck pin on status

#define PP_SEND 0b100000
#define PP_READ 0b000000

10

    unsigned 4 control_port;
    unsigned 6 status_port;
    unsigned 1 pp_dir with {warn = 0};
15    unsigned 2 command;
    unsigned 8 temp_data;

    PpSetStatus(PP_READ | FRTC); // initialise the port, read mode, FRTC high

20    while(1)
    {
        prialt
        {
            case pp_command ? command:

25

                // deal with any commands received
                switch (command)
                {
                    case OPEN_CHANNEL:
```

```

// open channel and set to FPGA send
mode

5      PpSetStatus(PP_SEND | FCC ); // |FDC
      keep FCC low, FRTC low to indicate ready
      pp_dir = 1;

10     // wait for pulse on HCC in response to
      open channel

      PpReadControl(control_port);

15     while(HCC) // wait for nHCC to go low
      {
          PpReadControl(control_port);

20     }

      while(!HCC) // wait for nHCC to go high
      {

25     PpReadControl(control_port);

      }

```



```
pp_error ! PP_NO_ERROR;
```

```
break;
```

5

```
case CLOSE_CHANNEL: // closes the channel
```

regardless of state

```
PpSetStatus(PP_READ | FRTC); // sets
```

10 status port to all zeros, FRTC high

```
pp_dir = 0;
```

```
pp_error ! PP_NO_ERROR;
```

```
break;
```

15

```
case SEND_MODE:
```

```
PpReadControl(control_port);
```

20

```
// set FRTC high - host send, start driving
```

data pins, FCC low

```
PpSetStatus(PP_SEND);
```

25

```
pp_dir = 1;
```

```
pp_error ! PP_NO_ERROR;
```

```
// BUFFERNOTFINISHED
```

```
break;
```

case RECV_MODE:

5 // set FRTC high - host read - stop driving

data pins, FCC high, FDC low

PpSetStatus(PP_READ | FCC);

//|FDC|FCC

pp_dir = 0;

10 pp_error ! PP_NO_ERROR ;

break;

15 default:

delay;

break;

}

20 break;

// FPGA sending

25 case pp_send_chan ? temp_data:

PpSetStatus(PP_SEND); // FCC low, FDC

low - pin is inverted

```

PpReadControl(control_port);

while(!HCC) // wait for host to de-assert

5  HCC

{
    PpReadControl(control_port);
}

10  PpWriteData(temp_data);
    PpSetStatus(PP_SEND | FDC); // FCC low,

    FDC high

    PpReadControl(control_port);

15

while(!HDC) // wait for host to assert HDC
{
    PpReadControl(control_port);

20
}

PpSetStatus(PP_SEND); // FCC low, FDC

low - pin is inverted

25

PpReadControl(control_port);

while(HDC) // wait for host to de-assert

HDC
```

```

    {
        PpReadControl(control_port);
    }

5      break;

        // host sending
    default:

10      PpReadControl(control_port);
        PpReadStatus(status_port);

15      if (!status_port[5] & !HCC) // read one
        byte, if in read mode and HCC is low
        {

20      while(!HDC) // wait for host to
        apply data and raise HDC
        {

        PpReadControl(control_port);

25      }
    }
```

```
FDC); // FCC high FDC high
```

```
PpSetStatus( PP_READ | FCC |
```

```
PpReadData(temp_data);
```

```
pp_rcv_chan ! temp_data;
```

```
PpReadControl(control_port);
```

```
PpReadStatus(status_port);
```

```
while(HDC) // wait for host to
```

```
{
```

```
remove HDC
```

```
PpReadControl(control_port);
```

```
}
```

```
PpSetStatus( PP_READ | FCC ); //
```

```
FCC high FDC low
```

```
}
```

```
else delay;
```

```
break;
```

```
}
```

```
} // while(1)
```

```
        delay; // avoid combinational cycles
    }

```

5

```
10  //////////////////////////////////////
    // Parallel Port - Physical layer
    //
    // Allows access to all the data, control and status ports
    // through a series of channels which can be read from
15  // and written to.
    //////////////////////////////////////

```

```
    // Macro abstractions for the various actions

```

```
20  macro proc PpWriteData(/*(unsigned 8)*/ byte)
    {
        pp_data_send_channel ! byte;

    }

```

25

```
macro proc PpReadData(/*(unsigned 8)*/ byte)
{
    pp_data_read_channel ? byte;

```

}

macro proc PpReadControl(/*(unsigned 4)*/ control_port)

5 {
 pp_control_port_read ? control_port;

}

10

macro proc PpReadStatus(/*(unsigned 6)*/ status_port)

{
 pp_status_port_read ? status_port;

15 }

macro proc PpSetStatus(/*(unsigned 6)*/ status_port)

{
 pp_status_port_write ! status_port;

20 }

25 // Actual Parallel Port control circuitry

macro proc parallel_port(pp_data_send_channel, pp_data_read_channel,
 pp_control_port_read,

```

                                pp_status_port_read,
pp_status_port_write)
{

5      unsigned 8 pp_data;
      unsigned 6 status_register;

      interface bus_ts_clock_in (unsigned 8) data_bus(pp_data, status_register[5])
with pp_data_pins;

10

      // Control Port (unsigned 4, made up as nSelect_in.in @ init.in @ nAutofeed.in
      @ nStrobe.in)
      interface bus_clock_in (unsigned 4) control_port() with control_port_pins;

15

      // Status Port, status_register = pp_direction @ busy @ nAck @ pe @ Select @
      nError;
      interface bus_out() status_port_bus(status_register[4:0]) with status_port_pins;

20

      // Setting pp_direction to 1 will drive data onto the pins.

      while(1)
      {

25          // Allows read of control, read / write of status and data ports
simulatneously
          par
          {

```



```
prialt
{
    case pp_control_port_read ! control_port.in:
        break;

    default:
        delay;
        break;
}
```

```
prialt
{
    case pp_status_port_write ? status_register:
        break;

    case pp_status_port_read ! status_register:
        break;

    default:
        delay;
        break;
}
```

```
prialt
{
    case pp_data_send_channel ? pp_data:
```

```
break;

case pp_data_read_channel ! data_bus.in:
    break;

5      default:
        delay;
        break;
    }

10      }

    }

15      delay; // to avoid combinational cycles
    }

20      //macro expr control_port = nSelect_in.in @ init.in @ nAutofeed.in @
nStrobe.in;

/*interface bus_clock_in (unsigned 1) nAutofeed() with nAutoFeed_pin;
25 interface bus_clock_in (unsigned 1) init() with init_pin;
interface bus_clock_in (unsigned 1) nSelect_in() with nSelect_in_pin;
interface bus_clock_in (unsigned 1) nStrobe() with nStrobe_pin;

// defined in the same order as on a PC
```

```
macro expr control_port = nSelect_in.in @ init.in @ nAutofeed.in @ nStrobe.in;

*/

/*
5   interface bus_out () nAck_line( status_register[3] ) with nAck_pin;
   interface bus_out () busy_line(status_register[4]) with busy_pin;
   interface bus_out () pe_line(status_register[2]) with pe_pin;
   interface bus_out () select_line(status_register[1]) with select_pin;
   interface bus_out () nError_line(status_register[0]) with nError_pin;
10  */

   // status_register[5] is high to send and low to receive
   // defined in the same order as on a PC
   //   macro expr status_port = pp_direction @ busy @ nAck @ pe @ Select @
15  nError;

20
```

Appendix D

This Appendix describes a Macro Library for a board according to the present invention. The library contains functions for

```
5      1) Memory arbitration
      2) Flash bus arbitration
      3) Read and Write to Flash RAM
      4) FPCOM settings
      5) Control of the LEDs

10
      //////////////////////////////////////
      //
      // Interfaces
      //
15  // Shared RAM arbitration
      // -----
      //   KRequestMemoryBank(bankMask)
      //   KReleaseMemoryBank(bankMask)
      //
20  //   Flash RAM Macros
      // -----
      //   KEnableFlash()
      //   KDisableFlash()
      //   KSetFlashAddress(address)
25  //   KWriteFlashData(address, data)
      //   KReadFlashData(address, data)
      //   KReadFlashID(flash_component_ID, manufacturer_ID)
      //
      //
```

```
//      Flash bus arbitration
//      -----
//      KSetFPGAFBM()
//      KReleaseFPGAFBM()
5 //
//      Others
//      -----
//      KSetLEDs(maskByte)
// KSetFPCOM(fpcom)
10

15 #ifndef _KOMPRESSOR_LIBRARY
#define _KOMPRESSOR_LIBRARY

// Include header file
#include "KompressorMaster.h"
20

////////////////////////////////////

// Request access to a memory bank
25 //
// The procedureS will block until access to all the requested banks have been
// granted.
//
```

```
unsigned 1 shared_bank0_request = 1 with { warn = 0} ;
```

```
unsigned 1 shared_bank1_request = 1 with { warn = 0} ;
```

```
interface bus_out() shbk0req(shared_bank0_request) with
```

```
5 sram_shared_bank0_request_pin;
```

```
interface bus_out() shbk1req(shared_bank1_request) with
```

```
sram_shared_bank1_request_pin;
```

```
interface bus_clock_in(unsigned 1) shbk0grant() with sram_shared_bank0_grant_pin;
```

```
interface bus_clock_in(unsigned 1) shbk1grant() with sram_shared_bank1_grant_pin;
```

```
10
```

```
macro proc KRequestMemoryBank0()
```

```
{
```

```
15     shared_bank0_request = 0;
```

```
    while(shbk0grant.in) delay;
```

```
}
```

```
20
```

```
macro proc KRequestMemoryBank1()
```

```
{
```

```
    shared_bank1_request = 0;
```

```
    while(shbk1grant.in) delay;
```

```
25 }
```

```
////////////////////////////////////
```

```
// Release a memory bank
```

```
//
```

5

```
macro proc KReleaseMemoryBank0()
```

```
{
```

```
    shared_bank0_request = 1;
```

```
}
```

10

```
macro proc KReleaseMemoryBank1()
```

```
{
```

```
    shared_bank1_request = 1;
```

15

```
}
```

20

```
////////////////////////////////////
```

```
//
```

25 // Functions for dealing with FP commands

```
#define FP_SET_IDLE          (unsigned 3)  7
```

```
#define FP_READ_STATUS (unsigned 3)  5
```

```
#define FP_CCLK_LOW          (unsigned 3)  3
```

Top secret - 4E52460

```
#define FP_CCLK_HIGH    (unsigned 3)  7
#define FP_WRITE_CONTROL (unsigned 3)  0
```

```
5  unsigned 3 fpcom = FP_SET_IDLE with { warn = 0}; // default
    interface bus_out() fpcom_bus(fpcom) with FPcom_pins;
```

```
macro proc KSetFPCOM(command)
```

```
{
```

```
10      fpcom = command;
        delay;
        delay;
```

```
}
```

```
15
```

```
macro proc KReadCPLDStatus(status)
```

```
{
```

```
    par
```

```
    {
```

```
20      KDisableFlash();
        flash_write = 0;
    }
```

```
    KSetFPCOM(FP_READ_STATUS);
```

```
25
```

```
        delay;
        delay;
        delay;
    delay;
```

FOR THE "F063T0" FAMILY


```
status = flash_data_bus.in;
```

```
par
```

```
{
```

```
5
```

```
    KSetFPCOM(FP_SET_IDLE);
```

```
    KEnableFlash();
```

```
}
```

```
}
```

```
10
```

```
macro proc KWriteCPLDControl(control)
```

```
{
```

```
    KDisableFlash();
```

```
    par
```

```
15
```

```
{
```

```
    flash_data = (unsigned 8) (0 @ control);
```

```
    flash_write = 1;
```

```
}
```

```
20
```

```
    KSetFPCOM(FP_WRITE_CONTROL);
```

```
    delay;
```

```
    delay;
```

```
    delay;
```

```
25
```

```
    par
```

```
{
```

```
        KSetFPCOM(FP_SET_IDLE);
```

```
        flash_write = 0;
```

```
        KEnableFlash();
```

```
    }
}

5  //////////////////////////////////////
   //
   //    Flash RAM stuff
   //
   //
10 // Parameters;
   //
   //    Read/write cycle          120ns
   //    Address to output         120ns
   //    CE to ouput               120ns
15 //
   //    CE low to WE low          0
   //    write pulse width low 70ns
   //    data setup to we high 50ns
   //    address setup to we hi 55ns
20 //    address/data hold         0ns
   //    write pulse width high 30ns

25 unsigned 24 flash_address with { warn = 0};
   unsigned 8 flash_data with { warn = 0};
   unsigned 1 flash_cs = 1, flash_we = 1, flash_oe = 1 with { warn = 0}; // initialise to
   high
```

unsigned 1 flash_write = 0 with { warn = 0}; // controls direction of the data pins

unsigned 1 flash_on = 0 with { warn = 0}; // controls the other tristate buses

interface bus_ts_clock_in(unsigned 24) flash_address_bus(flash_address, flash_on)

5 with {data = FA_pins};

interface bus_ts_clock_in(unsigned 1) flash_chipselect(flash_cs, flash_on) with

flash_cs_pin;

interface bus_ts_clock_in(unsigned 1) flash_writeenable(flash_we, flash_on) with

flash_we_pin;

10 interface bus_ts_clock_in(unsigned 1) flash_outputenable(flash_oe, flash_on) with

flash_oe_pin;

interface bus_ts_clock_in(unsigned 8) flash_data_bus(flash_data, flash_write) with

{data = FD_pins};

15

macro proc KEnableFlash()

{

par

{

20 flash_on = 1;

flash_cs = 0;

}

}

25

macro proc KDisableFlash()

{

par{

flash_on = 0;

```
flash_cs = 1;
}
}

5

// Sets up the address on the
macro proc KSetFlashAddress(address)
{
10     flash_address = address;
}

macro proc KWriteFlashData(address, data)
15 {

    par // set up address and data and drive onto pins
    {
        flash_oe = 1; // disable output
        flash_address = address;
        flash_data = data;
        flash_write = 1;
        flash_we = 0; // send write pulse
    }

25

    // running at 50/2 MHz - 40 ns cycles - 2 delays should be
    // sufficient to meet timing constraint

    delay;
```

delay;

par

{

5 flash_we = 1;
 flash_write = 1;
 }

}

10

macro proc KReadFlashData(address, data)

{

par

{

15 flash_write = 0;
 flash_oE = 0; // enable output
 flash_address = address;
 }

20 // running at 50/2 MHz - 40 ns cycles - 2 delays should be
 // sufficient to meet timing constraint

delay;

delay;

data = flash_data_bus.in;

25

}

macro proc KReadFlashID(flashid, manid)

```
{

    par
    {
5        KEnableFlash();
        KSetFPGAFBM();
    }

    KWriteFlashData(0, 0x90);
10    KReadFlashData(0, manid);
    KReadFlashData(2, flashid);

    par
    {
15    KReleaseFPGAFBM();
        KDisableFlash();
    }

20 }

macro proc KReadFlashStatus(status)
{
    par
25    {
        KEnableFlash();
        KSetFPGAFBM();
    }
}
```

```

    KWriteFlashData(0, 0x70);
    KReadFlashData(0, status);

    par
5      {
        KDisableFlash();
        KReleaseFPGAFBM();
      }

10  }

    //////////////////////////////////////
    // Flash bus arbitration pins
15  //
    unsigned 1 fbus_master = 1 with {warn = 0}; // initialise to not master
    interface bus_out() bus_master_line(fbus_master) with BUSMaster_pin;

    macro proc KSetFPGAFBM()
20  {
        fbus_master = 0;
    }

25  macro proc KReleaseFPGAFBM()
    {
        fbus_master = 1;
    }

```

```
////////////////////////////////////
// LED control macros

5

unsigned 8 LED = 0 with {warn = 0}; // by default
unsigned 1 LED_en = 0 with {warn = 0};
interface bus_ts(unsigned 8) LEDpins(LED, LED_en) with LED_pins;
10 macro proc KSetLEDs(maskByte)
    {
        par
            {
                LED = maskByte;
15         LED_en = 1;
            }
    }

20

////////////////////////////////////
//
// FPcom ==7 CCLK = High
//
25 // From the FPGA BUSMaster pin should be brought low and the FLASH may be
// accessed as any normal device RAM device.
//
#endif _KOMPRESSOR_LIBRARY
```